08-What is String

What is a String in Java?

Overview of Strings

A **String** is generally understood as a sequence of characters. In Java, however, a string is an object that represents a sequence of characters. The ***java.lang.String*** class is used to create and manipulate string objects.

- **Data Type**: Just as there are data types for storing variables (like int, char, etc.), String is a data type used to store a sequence of characters or words within double quotes ("").
- **Character Array**: You can also use a character array to store a series of characters.

Creating a String Object

There are two main ways to create a String object in Java:

1. **By String Literal**
2. **By Using the new Keyword**

1. String Literal

- A string literal is created by placing the characters within double quotes.

```
String s = "Navin";
```

- When a string literal is created, the JVM checks the **String Constant Pool**:
    - If the string already exists in the pool, a reference to the existing object is returned.
    - If it doesn't exist, a new string object is created and added to the pool.

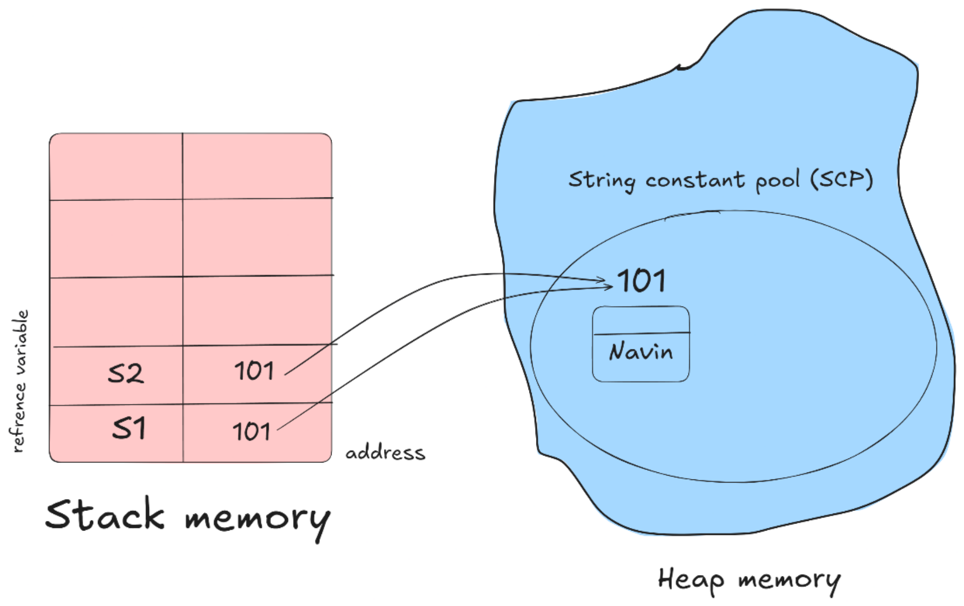
Example:

```
String s1 = "Navin";

String s2 = " Navin "; // Does not create a new object, reuses the one in the pool
```

- **Explanation**: Only one object is created in this case. The JVM first checks if " Navin " is already in the pool. If it is, both s1 and s2 will reference the same object.

TELUSKO



2. Using the new Keyword

- Strings can also be created using the new keyword:

```
String s = new String("Navin ");
```

- This approach creates two objects:
  - One in the heap memory (outside the string constant pool).
  - Another in the string constant pool.

Example:

```
String name = new String(); // Creates an empty String object
```

- **Explanation**: name is a reference variable that points to the String object. This syntax, however, is less commonly used.

Internal Architecture

- **Concatenation**: You can concatenate two strings using the + operator.

Example:

```
String name = new String("Navin");

System.out.println("Hello " + name);
```

Output:

```
Hello Navin
```

String Methods

The String class provides several methods to perform various operations on strings. Here are some of the most commonly used methods with examples:

1. **length()**: Returns the length of the string.

```
String str = "Hello";
int len = str.length(); // Returns 5
```

2. **concat(String s)**: Concatenates the specified string to the end of the current string.

```
String s1 = "Hello";
String s2 = "World";
String s3 = s1.concat(s2); // Returns "HelloWorld"
```

3. **equals(Object obj)**: Compares the string to the specified object for equality.

```
String s1 = "Hello";
String s2 = "Hello";
boolean isEqual = s1.equals(s2); // Returns true
```

4. **substring(int beginIndex)**: Returns a new string that is a substring of this string.

```
String str = "HelloWorld";
String subStr = str.substring(5); // Returns "World"
```

5. **replace(char oldChar, char newChar)**: Replaces occurrences of the old character with the new character.

```
String str = "Hello";
String replacedStr = str.replace('l', 'p'); // Returns "Heppo"
```

6. **split(String regex)**: Splits the string around matches of the given regular expression.

```
String str = "one,two,three";
String[] parts = str.split(","); // Splits into ["one", "two", "three"]
```

7. **compareTo(String anotherString)**: Compares two strings lexicographically.

```
String s1 = "Apple";
String s2 = "Banana";
int result = s1.compareTo(s2); // Returns a negative value because "Apple" is less than
"Banana"
```

8. **intern()**: Returns a canonical representation for the string object.

```
String s = new String("Hello");
String internedStr = s.intern(); // Interns the string
```